VOLUME FIFTEEN

ANNUAL REPORTS IN COMPUTATIONAL CHEMISTRY

Edited by

DAVID A. DIXON
The University of Alabama, Tuscaloosa, AL, United States

ELSEVIER

CHAPTER FOUR

Tensor representations and symmetry in many-electron wave functions

T. Daniel Crawford[a,b,*], **Roberto Di Remigio**[a,c]

[a]Department of Chemistry, Virginia Tech, Blacksburg, VA, United States
[b]Molecular Sciences Software Institute, Blacksburg, VA, United States
[c]Hylleraas Centre for Quantum Molecular Sciences, Department of Chemistry, University of Tromsø—The Arctic University of Norway, Tromsø, Norway
*Corresponding author: e-mail address: crawdad@vt.edu

Contents

1. Introduction 79
2. Basics of tensor storage 81
3. Tensor symmetries 85
   3.1 Permutational symmetry 85
   3.2 Spin symmetry 86
   3.3 Spatial/point group symmetry 87
4. Modern implementations 94
5. Conclusions and prospectus 98
Acknowledgments 99
References 99

Abstract

We discuss from a pedagogical perspective the use of tensors in many-body electronic structure methods, especially the relevant storage and computational aspects used by modern quantum chemistry software packages. We consider the implementational consequences of the various symmetries—spin, spatial, and permutational—that appear in tensors representing the Hamiltonian, wave functions, and other important quantities in many-body methods. In addition, we review a number of state-of-the-art approaches to tensor frameworks on modern high-performance computing architectures.

1. Introduction

Quantum chemical models are typically formulated in terms of basis-set expansions, and the linear- and nonlinear equations governing these models are thus most conveniently expressed as contractions over tensor-based

Annual Reports in Computational Chemistry, Volume 15
ISSN 1574-1400
https://doi.org/10.1016/bs.arcc.2019.08.005

representations of the corresponding Hamiltonian integrals and wave function parameters. While two-electron repulsion integrals require up to four-dimensional tensors, the complexity of the data structures for the wave function coefficients depends on the complexity of the electronic structure model. Self-consistent-field methods such as Hartree–Fock or density-functional theory require only two-dimensional tensors (matrices) for their molecular orbital coefficients or electron densities, whereas more advanced electron correlation methods, such as coupled cluster theory, can require as high as $2N$-dimensional tensors, where N is the level of excitation in the wave function ansatz. For example, the coupled cluster singles and doubles (CCSD) method requires up to four-dimensional tensors for the storage of the cluster amplitudes, while the inclusion of full triple excitations (CCSDT) requires up to six-dimensional tensors.

The performance of computer implementations for solving the complicated algebraic equations underlying such methods thus hinges on the details of the tensor representation, and the choice of data layout/distribution, incorporation of permutational, spatial, and/or spin symmetries, etc. is often tied to the particular choice of high-performance computing hardware on which the program is ultimately deployed. The large storage capacities of distributed-memory computing architectures, for example, permit calculations involving much larger tensors than single-node computers, but they also require much greater attention to the design of efficient tensor data layouts in order to minimize internode communications.

In this article, we will discuss the details of a number of tensor representations widely used in electronic structure codes, as well as their advantages and disadvantages for specific applications. We will take a pedagogical approach to our presentation in order to appeal primarily to newcomers to the field, as opposed to "old hands" who are likely already adept at many of the concepts we consider. Given the wide array of computer languages utilized in modern quantum chemistry software (Fortran, C, C++, Python, etc.) we will avoid any language-specific examples. While some of the techniques we describe here can be found scattered throughout the computational chemistry literature, others can be found only within the codes themselves and have not been previously published in detail. Finally, where appropriate we will give specific examples of community codes that utilize these methods, but our overview is not intended to be fully comprehensive of the current software ecosystem of the field. Thus, the lack of recognition of any particular codes in our presentation should not be taken as a form of *expressio unius est exclusio alterius*.

2. Basics of tensor storage

Modern high-performance computers are particularly adept at algorithms involving products of matrices, both because of the structure of advanced CPU architectures (pipelining of low-level instructions, multi-level cache memory, fused multiply-add/-accumulate operations) and the availability of highy tuned algorithms to exploit these capabilities. Thus, all advanced quantum chemistry programs are designed to take advantage of matrix-based data structures. Highly optimized libraries of matrix mathematics functions, such as the basic linear algebra subroutines (BLAS), are available for most architectures and operating systems. Such libraries are capable of providing near-peak performance of matrix operations in single-core computing systems, and standard interfaces are available that allow the implementation of efficient electronic structure programs without dependence on a particular matrix-algebra package.

As an example of the use of matrix algorithms, consider the following, typical tensor contraction, written in Einstein notation in which summation is implied over repeated indices:

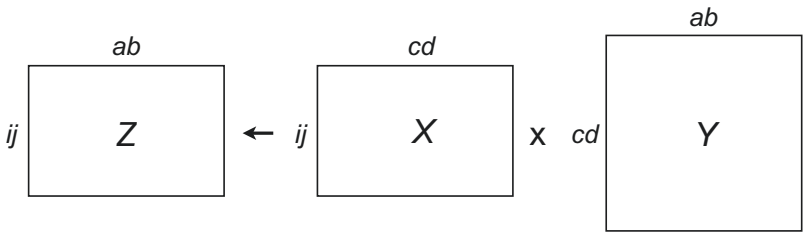
$$z_{ij}^{ab} = x_{ij}^{cd} y_{cd}^{ab}, \tag{1}$$

we will assume for now that these tensors exhibit no special symmetries (e.g., permutational symmetry). A naïve implementation would store each tensor as a four-dimensional array and carry out the multiplications and additions explicitly. However, a more efficient implementation would be to pack each tensor into rectangular matrices and utilize optimized linear algebra libraries. To make this example more concrete but simple, assume that indices i and j range from 0 to 2 and indices a, b, c, and d range from 0 to 3. Then, for example, we can arrange the $3 \times 3 = 9$ possible combinations of i and j as

i	*j*	*ij*
0	0	0
0	1	1
0	2	2
1	0	3
1	1	4

1	2	5
2	0	6
2	1	7
2	2	8

where the compound index $ij = 3 \times i + j$ for a total of nine ij combinations. Assuming a comparable structure for the 14 possible combinations of corresponding ab and cd compound indices, we may view the three tensors x, y, and z as simple matrices and the contraction as a matrix-matrix product, depicted schematically as

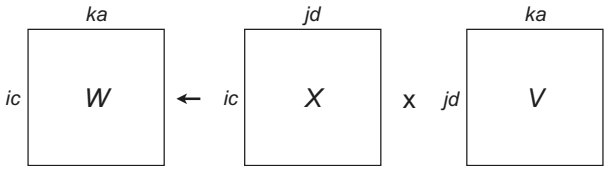


Thus, one may take advantage of the efficiency of matrix operations on modern computing hardware by arranging the tensors into such matrices using contiguous memory storage and passing pointers or references to the start of this memory to optimized BLAS functions, particularly the general matrix multiply (GEMM) routines. In most quantum chemical programs, double-precision (64-bit) storage and multiplication is used for such tensors, but recent work has suggested that single precision (32-bit) can provide better performance without loss of necessary precision. We emphasize that the storage of such tensors must be contiguous in memory (i.e., each consecutive element of the matrix must occupy consecutive elements of memory) in order to conform to the standard BLAS implementations. This implies that one must take care when allocating memory dynamically for storing such tensors. In C/C++ programs, the default storage is *row-wise*, meaning that the beginning of a row of the matrix immediately follows the end of the preceding row in linear memory. However, the default interface in many BLAS implementations is Fortran based, which assumes that the default storage is *column-wise*, and thus many codes will implicitly transpose and reorder the matrix-matrix multiplication to account for this without the need to sort the tensor elements. There do exist C/C++-based BLAS interfaces that provide row-wise access, and these are gaining wider adoption.

In many electronic structure methods, such as coupled cluster, the same tensor may appear in multiple contractions, but with different index orderings. For example, the x_{ij}^{cd} tensor above might appear in another contraction as

$$w_{ik}^{ca} = x_{ij}^{cd} v_{jk}^{da}. \tag{2}$$

In this case, the matrix-based storage used for the previous contraction no longer applies, because the row and column indices have been changed, as shown in the schematic below:

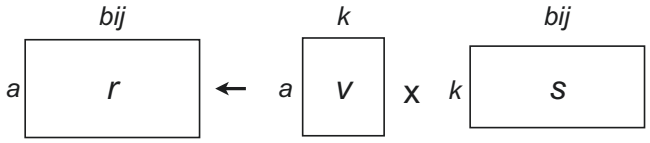


In order to take advantage of the efficient matrix-matrix multiplication algorithms of the BLAS, the x_{ij}^{cd} tensor (as well as w_{ik}^{ca} and v_{jk}^{da}, if they are used in other contractions) must first be sorted into the appropriate element ordering, and many production-level programs may keep several orderings of a given tensor simultaneously, provided sufficient memory and other storage.

A perhaps subtle point, however, is that not all tensor-index reorderings require explicit movement of data. Consider yet another contraction of the form

$$r_{ij}^{ab} = v_k^a s_{kb}^{ij}. \tag{3}$$

In this case we have a contraction of a two-index tensor and a four-index tensor to yield another four-index tensor, with only a single summation index (k). To take advantage of the BLAS matrix-matrix multiplication algorithm, we require the tensor ordering shown schematically below:



Thus, it appears that we must ensure that the elements of the s_{kb}^{ij} tensor are ordered such that only k stands as the row index against a compound bij index for the columns. If we have already stored the elements of s_{kb}^{ij} in a matrix with a kb compound row index and a ij compound column index, it appears we must sort the elements into the new arrangement. However, closer inspection reveals that no movement of data is actually required.

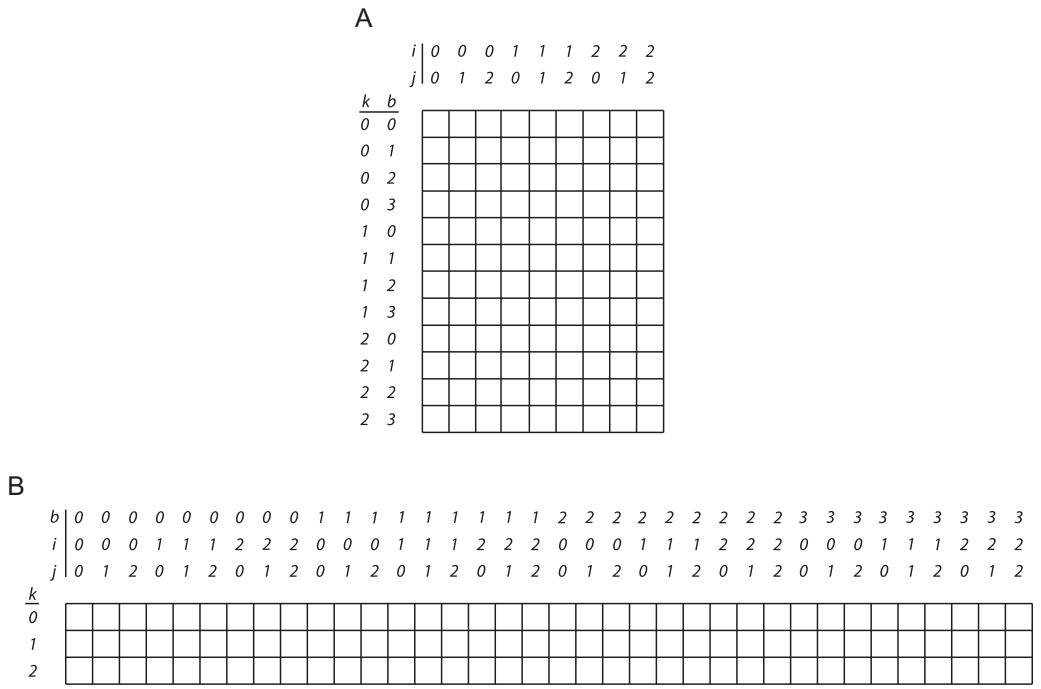


Fig. 1 An example of matrix-based storage of the s^{ij}_{kb} tensor for a system with $i, j, k \in \{0, 1, 2\}$ and $b \in \{0, 1, 2, 3\}$ orbitals. In (A), the tensor elements are arranged with compound row and column indices of *kb* and *ij*, respectively, while in (B) the rows are labeled by only *k* while the columns use the compound index *bij*. If row-wise contiguous memory storage is used, the two arrangements are, in fact, identical.

Assuming that the indices i, j, and k, range from 0 to 2 and the indices a and b range from 0 to 3, as before, then a $kb \times ij$ matrix-based storage of s^{ij}_{kb} corresponds to the arrangement shown in Fig. 1A, where we have explicitly indicated the values of the individual indices corresponding to each element of the 12×9 matrix. If the matrix is stored row-wise, then the first element in memory corresponds to s^{00}_{00}, followed by s^{01}_{00}, s^{02}_{00}, etc. The last element of the first row corresponds to s^{22}_{00}, and, if the rows are stored contiguously in memory, the next value is the first element of the second row, i.e., s^{00}_{01}.

Compare this ordering to that required to carry out the contraction in Eq. (3) above, which is shown in Fig. 1B as a 3×36 matrix. Again the first element is s^{00}_{00}, followed by s^{01}_{00}, s^{02}_{00}, etc. But in this case, the last element of the first row is s^{22}_{03}, and it is followed by the first element of the next row, *viz*. s^{00}_{10}. However, the first nine tensor elements end with s^{22}_{00}, followed by s^{00}_{01}, just as occurred in the $kb \times ij$ ordering of the tensor. Therefore, a shift between the two arrangements requires no movement of the data in memory, only perhaps a recalculation of the pointers to the beginning of each row in memory (but this is not necessary for the BLAS matrix-matrix multiplication

algorithm). This observation also holds if we required the matrix to exhibit a $kbi \times j$ arrangement, assuming no packing of the i and j indices, e.g., to take advantage of permutational symmetry (*vide infra*) — no movement of the tensor elements in memory is needed. Furthermore, this observation holds for tensors of any order: given an n-dimensional tensor with elements stored contiguously in linear memory, any partitioning of the dimensions along adjacent indices requires no movement of tensor elements within memory. A six-dimensional tensor, for example, $x(i, j, k, a, b, c)$, could be viewed equivalently as a matrix with index dimensions $i \times jkabc$, $ij \times kabc$, $ijk \times abc$, $ijka \times bc$, or $ijkab \times c$.

3. Tensor symmetries

Electronic structure calculations can take advantage of a wide array of wave function symmetries to streamline both the storage requirements and floating-point operational costs of a calculation, including permutational, spin, and spatial symmetries. Here we summarize the basic concepts behind utilizing these symmetries and what advantages and disadvantages they provide.

3.1 Permutational symmetry

The Pauli antisymmetry principle indicates that wave functions describing fermions (which includes electrons) must change sign upon the interchange of the (spin and spatial) coordinates of any two particle. This principle is often manifested in electronic structure calculations in the antisymmetry of the various tensors that appear in the second-quantized representation of the Hamiltonian and the electronic wave function. For example, the two-electron integrals are often expressed in Dirac/physicist notation as

$$\langle pq \,\|\, rs\rangle = \int d\vec{x}_1 \int d\vec{x}_2 \phi_p^*(\vec{x}_1)\phi_q^*(\vec{x}_2)\frac{1}{r_{12}}\big(\phi_r(\vec{x}_1)\phi_s(\vec{x}_2) - \phi_s(\vec{x}_1)\phi_r(\vec{x}_2)\big), \tag{4}$$

where $\vec{x}_1$ and $\vec{x}_2$ contain the spin (ω) and spatial ($\vec{r}$) coordinates of each electron, r_{12} is the length of difference in the space vectors of the two electrons, and the $\{\phi_p\}$ denote spin orbitals. This definition bestows on this tensor the antisymmetry:

$$\langle pq \,\|\, rs\rangle = -\langle pq \,\|\, sr\rangle = -\langle qp \,\|\, rs\rangle = \langle qp \,\|\, sr\rangle, \tag{5}$$

and, if the spin orbitals are real functions,

$$\langle pq \,\|\, rs\rangle = \langle rs \,\|\, pq\rangle = -\langle rs \,\|\, qp\rangle = -\langle sr \,\|\, pq\rangle = \langle sr \,\|\, qp\rangle. \tag{6}$$

When the Hamiltonian is formulated as above, the wave function amplitudes in electron correlation methods also exhibit antisymmetry, e.g., the double-excitation amplitude tensor of coupled cluster theory,

$$t_{ij}^{ab} = -t_{ij}^{ba} = -t_{ji}^{ab} = t_{ji}^{ba}, \tag{7}$$

where i and j (a and b) denote occupied (unoccupied) spin orbitals.

One can take advantage of permutational antisymmetry by packing the canonical indices described earlier. For example, if the tensor x_{ij}^{cd} of Eq. (1) is antisymmetric with respect to permutation of its i and j indices, then the earlier table of values of the compound index ij is reduced from nine elements to only three because tensor elements with $i = j$ are necessarily zero:

i	*j*	*ij*
1	0	0
2	0	1
2	1	2

While this results in considerable reduction of the storage and computing demands, a complication arises when the tensor indices must be unpacked for contractions that require all values of the permutable indices, such as that occurring in Eq. (2).

3.2 Spin symmetry

In the case of a spin-independent Hamiltonian, the total spin angular momentum operator ($\hat{S}^2$), commutes with the Hamiltonian and thus eigenfunctions can be constructed that are common to both operators. In practice, this typically means that approximate wave functions are constructed to be eigenfunctions of $\hat{S}^2$, i.e., to represent a pure spin state. This is relatively straightforward for Hartree–Fock wave functions and for wave functions for which the wave operator is linear in nature, such as in configuration interaction methods. For more complicated electron correlation models, such as coupled cluster theory, enforcing spin symmetry is straightforward

only for states whose zeroth-order wave function represents a closed-shell singlet. For open-shell states, conventional coupled cluster approaches do not yield spin eigenfunctions unless one takes great pains in both formulation and implementation [1, 2].

While spin-orbital representations, such as those described in the previous subsection, yield antisymmetric tensors for Hamiltonian components and wave function parameters, the tensors representing spin-adapted wave functions are typically spin free (i.e., constructed in terms of spatial orbitals rather than spin orbitals), and thus have more limited permutational symmetry. For example, in a spin-adapted representation of a spin singlet, the coupled cluster double-excitation amplitudes mentioned earlier have only one permutational degree of freedom,

$$t_{ij}^{ab} = t_{ji}^{ba}, \tag{8}$$

where the indices here refer to spatial orbitals rather than spin orbitals.

3.3 Spatial/point group symmetry

While it is certainly true that the vast majority of molecules are asymmetric, the existence of even one symmetry element[a] can make a significant difference in the cost of an electronic structure calculation. The use of spatial symmetry to streamline quantum chemical calculations hinges on the vanishing integral rule, which states in its simplest terms that an integral of a general function over a symmetric domain is zero unless the integrand contains a component that transforms as the totally symmetric irreducible representation (irrep) of the applicable point group. While this rule holds for any point group, irreps, it is most straightforwardly derived for Abelian groups, for which all irreps are one-dimensional.

Consider such a point group comprised of h symmetry operations, $\hat{O}$ and, correspondingly h one-dimensional irreps with characters $\chi(\hat{O})$. An arbitrary function, $f(\vec{x})$, defined on a domain $\mathcal{R}$, may be written as a sum of components, each spanning an irrep of the point group,

$$f(\vec{x}) = \sum_{j}^{\text{irreps}} f(\vec{x})_j, \tag{9}$$

[a] We focus here on point group symmetry, which is of principal importance to molecular systems, as opposed to space-group symmetry, which is relevant to periodic systems, such as molecular crystals.

where we will take $j = 1$ to denote the totally symmetric irrep. Thus, the integral of $f(\vec{x})$ over $\mathcal{R}$ may similarly be decomposed into irrep components,

$$I = \int_{\mathcal{R}} f(\vec{x}) d\vec{x} = \sum_{j}^{\text{irreps}} \int_{\mathcal{R}} f(\vec{x})_j d\vec{x} = \sum_{j}^{\text{irreps}} I_j \tag{10}$$

Furthermore, the action of a symmetry operator $\hat{O}$ on the components of $f(\vec{x})$ yields the corresponding characters as eigenvalues, *viz.*

$$\hat{O} \sum_{j}^{\text{irreps}} f(\vec{x})_j = \sum_{j}^{\text{irreps}} \chi(\hat{O})_j f(\vec{x})_j, \tag{11}$$

and therefore,

$$\hat{O} I = \hat{O} \sum_{j}^{\text{irreps}} I_j = \sum_{j}^{\text{irreps}} \chi(\hat{O})_j I_j. \tag{12}$$

If the domain $\mathcal{R}$ is unchanged by the operations of the group, $\hat{O}[\mathcal{R}] \rightarrow \mathcal{R}$, then the integral of $f(\vec{x})$ over the domain $\mathcal{R}$ is concomitantly unchanged. Thus,

$$\hat{O} I - I = \sum_{j}^{\text{irreps}} \chi(\hat{O})_j I_j - \sum_{j}^{\text{irreps}} I_j = 0. \tag{13}$$

Given that this equation holds for all operations of the group, we may therefore sum over all such operations and reorder the summations to obtain,

$$\sum_{\hat{O}} \left(\hat{O} I - I \right) = \sum_{j}^{\text{irreps}} I_j \sum_{\hat{O}} \left(\chi(\hat{O})_j - 1 \right) = 0. \tag{14}$$

For point groups with nondegenerate irreps [3],

$$\sum_{\hat{O}} \chi(\hat{O})_j = \begin{cases} h \text{ if } j = \text{totally symmetric irrep} \\ 0 \text{ otherwise} \end{cases} \tag{15}$$

Thus,

$$\sum_{\hat{O}} \left(\hat{O} I - I \right) = \sum_{j \neq 1}^{\text{irreps}} I_j \sum_{\hat{O}} (-1) = -h \sum_{j \neq 1}^{\text{irreps}} I_j = 0. \tag{16}$$

We therefore conclude that the sum of the nontotally symmetric integrals is zero and that only the integral containing the totally symmetric irrep can make a nonzero contribution to the total integral, i.e.,

$$I = I_1. \tag{17}$$

The implication of the vanishing integral rule in electronic structure calculations is straightforward: if the one- and two-electron integrals appearing in the second-quantized Hamiltonian are comprised of molecular orbitals that are themselves built from symmetry-adapted basis functions (a common procedure in modern quantum chemical programs), then the integrals — and, by extension, the molecular orbitals and other wave function parameters/amplitudes and any intermediate tensors built from their products — are zero unless the direct product of their corresponding irreps contains the totally symmetric irrep. This result is probably the most fundamental selection rule in computational chemistry, as it shows that the matrix representation of a totally symmetric operator is block diagonal by irrep in a basis of symmetry-adapted functions. Hence the storage requirements for such matrix representations can be dramatically reduced as only the diagonal blocks will be nonzero. In the specific case of the antisymmetrized two-electron integral in Eq. (4), for example, if the molecular orbitals, $\boldsymbol{\phi}_p$, $\boldsymbol{\phi}_q$, $\boldsymbol{\phi}_r$, and $\boldsymbol{\phi}_s$ transform as the irreducible representations (*irreps*) Γ_p, Γ_q, Γ_r, and Γ_s, respectively, then the antisymmetrized two-electron integral, $\langle pq \,\|\, rs\rangle$, must be zero unless

$$A_1 \subset \Gamma_p \otimes \Gamma_q \otimes \Gamma_r \otimes \Gamma_s, \tag{18}$$

where we have used the Mulliken symbol A_1 to represent the totally symmetric irrep.

Most quantum chemistry programs can take advantage only of point groups containing nondegenerate irreps, specifically D_{2h} and its subgroups (C_{2v}, C_{2h}, C_2, D_2, C_s, C_i, and C_1), for which all characters are ± 1. (A notable exception is the SCF module in the Turbomole package [4].) As a result, direct products between the irreps of such groups are much easier to compute than for groups with complex characters, such as the cyclic groups C_3 or higher, or groups with degenerate irreps, where direct products can lead to direct sums of irreps (e.g., $e' \otimes e' = a_1 \oplus a_2 \oplus e'$ in the C_{3v} group). Furthermore, as first demonstrated in 1974 by Almlöf [5, 6], for D_{2h} and its subgroups direct products can be computed trivially and efficiently using the

bitwise exclusive-or (XOR) logical operation.[b] For example, for D_{2h}, if we use the irrep ordering of Cotton [3] and number them starting from 0 with the totally symmetric irrep first, we have

Irrep	Decimal	Binary
a_g	0	000
b_{1g}	1	001
b_{2g}	2	010
b_{3g}	3	011
a_u	4	100
b_{1u}	5	101
b_{2u}	6	110
b_{3u}	7	111

Then we may take any direct product without consulting a character table simply by computing the XOR between the binary representations of the two irreps in question, e.g.,

$$b_{1g} \otimes a_u = \mathrm{XOR}(001, 100) = 101 = 5 = b_{1u} \tag{19}$$

or

$$b_{2u} \otimes b_{3u} = \mathrm{XOR}(110, 111) = 001 = 1 = b_{1g}. \tag{20}$$

The XOR approach also maintains the requirement that, for D_{2h} and its subgroups, the direct product of any irrep with itself must yield the totally symmetric irrep, because XOR of any bit with itself is zero, e.g.,

$$b_{2g} \otimes b_{2g} = \mathrm{XOR}(010, 010) = 000 = 0 = a_g. \tag{21}$$

This technique has long been employed in the DALTON [7] and PSI [8] programs (and likely others) and XOR is just one of many examples in which bitwise logic operators have found use in efficient quantum chemical programs [5, 6].

We may simultaneously take advantage of both the vanishing integral rule and the matrix-based algorithms described in the previous section by

[b] The bitwise XOR can be thought of as "one or the other, but not both," and thus gives the single-bit values XOR(0,0) = XOR(1,1) = 0, and XOR(0,1) = XOR(1,0) = 1.

grouping the molecular orbitals by irrep. Taking the two irreps of the C_s point group as an example, assume that the three occupied orbitals in our earlier example transform such that the first two (orbitals 0 and 1) belong to the a' irrep and the last (orbital 2) belongs to a''. Similarly, let us assume that the four unoccupied orbitals are evenly divided with orbitals 0 and 1 in a' and orbitals 2 and 3 in a''. With these symmetry assignments in mind, we may augment the matrix storage for the s^{ij}_{kb} tensor used as an example from the previous section as shown in Fig. 2A to include the direct product of the irreps of each row pair, $\Gamma_{kb} \equiv \Gamma_k \otimes \Gamma_b$, and column pair, $\Gamma_{ij} \equiv \Gamma_i \otimes \Gamma_j$. Given that the tensor represents a totally symmetric quantity, then, in accord with Eq. (18), the only possible nonzero entries in the matrix are those for which $\Gamma_{kb} = \Gamma_{ij}$. In the figure, dark squares represent symmetry-allowed tensor elements and the white squares represent zeroes, revealing substantial sparsity in the matrix, with exactly half of the 108 possible tensor elements vanishing due to symmetry. If we were to sort the rows and columns to collect all

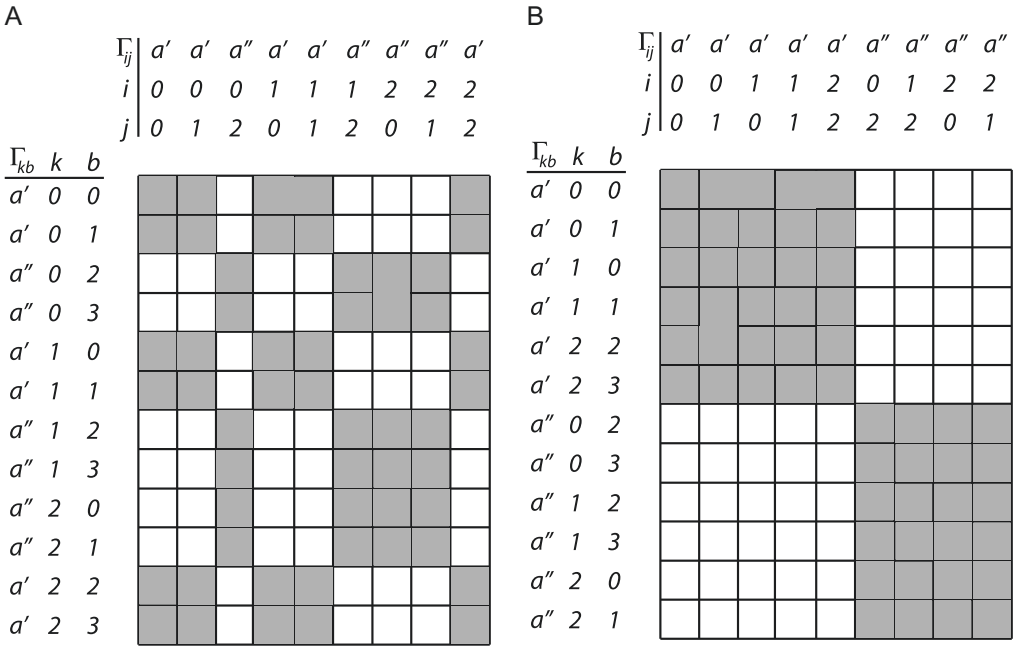
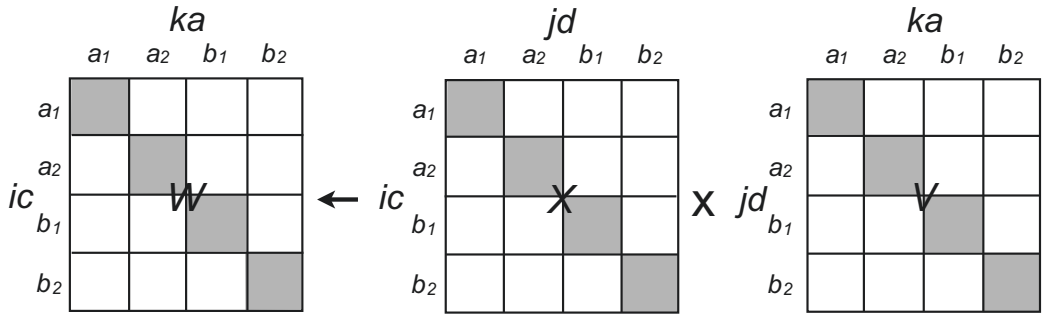


Fig. 2 An example of matrix-based storage of the s^{ij}_{kb} tensor, including point group symmetry. The molecule belongs to the C_s point group with the three occupied orbitals distributed between the irreps as $\{0, 1, 2\} = \{a', a', a''\}$ and the four unoccupied orbitals as $\{0, 1, 2, 3\} = \{a', a', a'', a''\}$. *Dark squares* indicate symmetry-allowed elements and *white squares* indicate vanishing elements. In (A) the rows and columns are ordered exactly as in Fig. 1, while in (B) *kb* and *ij* pairs have been sorted such that all binary products of orbitals yielding a' and a'' are grouped separately, leading to a more efficient symmetry-blocked structure.

a' and a'' irrep products together, we arrive at the blocked structure indicated in Fig. 2B. Thus, we may reduce the memory and/or disk requirements for storing the elements by a factor of two by retaining only the two symmetry-allowed submatrices of dimensions 6×5 and 6×4, respectively. In general, the reduction in storage costs of elements, amplitudes, and other symmetry-adapted intermediates scales approximately as the *square* order of the point group, h^2, with the eight symmetry operations of the D_{2h} group yielding an impressive factor of 64 reduction [9, 10].

Note also that the storage of the s_{kb}^{ij} tensor from the $kb \times ij$ arrangement shown in Fig. 2B may also be viewed as equivalent to a $k \times bij$ arrangement analogous to that in Fig. 1B, with the additional caveat that each nonvanishing submatrix is subsequently decomposed into h blocks of elements. For example, consider the upper-left submatrix of $kb \times ij$ elements in Fig. 2B, for which $\Gamma_{kb} = \Gamma_{ij} = a'$. If we isolate the index, k, to the left, we see that the two a' occupied orbitals can be grouped together, followed by the a'' occupied orbital. Therefore, this $\Gamma_{kb} = \Gamma_{ij} = a'$ submatrix may be viewed as two consecutive matrices, one with $\Gamma_k = \Gamma_{bij} = a'$ and one with $\Gamma_k = \Gamma_{bij} = a''$, with no movement of the elements in memory, as shown in Fig. 3A. Similarly, the $\Gamma_{kb} = \Gamma_{ij} = a''$ submatrix of Fig. 2B may be viewed as two submatrices, each distinguished by again isolating the index k to left within each irrep, as shown in Fig. 3B.

The use of symmetry not only reduces disk and memory storage requirements, it also reduces the number of floating-point operations required to evaluate tensor contractions appearing in electronic structure formulations. For example, for a molecule with C_{2v} symmetry, if we were to make use of symmetry-blocking of the tensors representing the x_{ij}^{cd}, v_{jk}^{da}, and w_{ik}^{ca} tensors from Eq. (2), the schematic diagram given earlier could be modified for the four irreps as



The irrep labels denote direct products of pairs of irreps associated with spin-adapted indices, Γ_{ic}, Γ_{jd}, or Γ_{ka}, and thus the dark squares indicate groups of symmetry-allowed elements and the white squares groups of

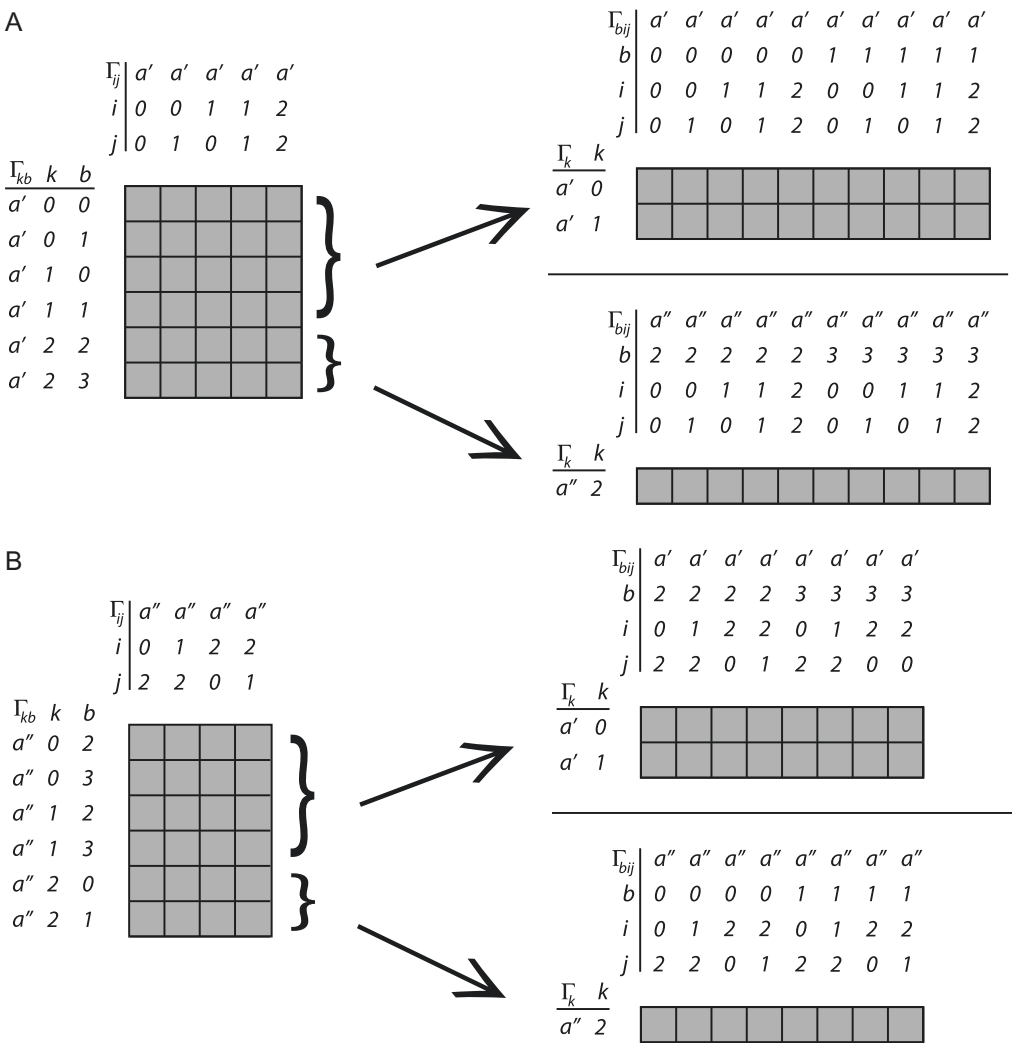
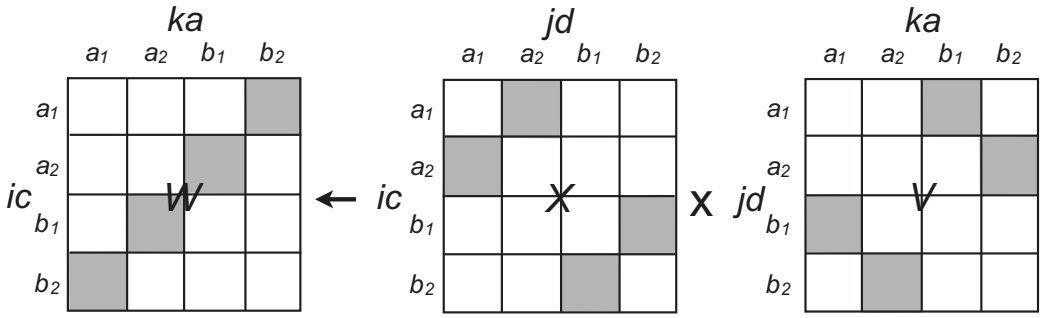


Fig. 3 Rearrangement of the s^{ij}_{kb} tensor into matrix-blocked $k \times bij$ ordering, including point group symmetry. The molecule belongs to the C_s point group with the same orbital groups as in Fig. 2. In (A) the $\Gamma_{kb} = \Gamma_{ij} = a'$ submatrix is decomposed into two smaller submatrices based on the two possible irreps of orbital index k, while in (B), the same decomposition is carried out on the $\Gamma_{kb} = \Gamma_{ij} = a''$ submatrix. In each case, if the tensor is stored in a row-wise, contiguous fashion in memory, no sorting of the data is necessary.

elements that must be zero.[c] The diagram reveals the useful factor of four ($= h$ for C_{2v}) reduction in storage of the residuals, amplitudes, and integrals, as well as the reduction in the number of floating-point operations by a factor of 16 ($= h^2$), because we only need carry out matrix-matrix multiplications between corresponding nonzero subblocks of the tensor factors.

[c] The perfect blocking in this schematic diagram in which each irrep contains the same number of tensor elements occurs only if the number of orbitals in each irrep is identical. This often occurs for cyclic groups, such as C_2 or for the inversion group, C_i, but not for groups with mirror planes, such as C_{2v}.

While the vanishing integral rule holds for the Hamiltonian and wave functions, some nonzero tensor quantities appearing in electronic structure methods may, in fact, be nontotally symmetric, i.e., transform as an irrep other than A_1.[d] For example, wave function derivatives with respect to a symmetric/symmetrized perturbation such as an external Cartesian field or symmetry-adapted geometrical coordinate can be nontotally symmetric, as can excitation vectors, e.g., tensors representing differences or gradients between different electronic states. In such cases, the schematic view of the contraction above takes on a somewhat different layout if, for example, the irrep of the x_{ij}^{cd} and v_{jk}^{da} factors were A_2 and B_1, respectively, within C_{2v}:



In this case, the same storage and computational advantages appear as in the totally symmetric case, but the implementation requires additional logic for the ordering of the nonzero data blocks. One could choose to order the submatrix storage based on either the row irrep or the column irrep, and, ultimately the decision is arbitrary, provided one maintains a consistent standard [8].

4. Modern implementations

Quantum chemistry is an integral part of the toolbox of researchers, in part because of the development of efficient and robust implementations of reliable *ab initio* methods that can leverage existing medium- and large-scale computer architecture. In addition, the formulation and application of new electronic structure methods that can enable new discoveries and new science critically depends on the ease with which quantum chemists can correctly and efficiently implement the equations inherent to these methods. The complexity of many of the highly accurate theories available makes this

[d] Such quantities still obey the vanishing integral rule, but they should be considered factors in a product of terms that ultimately must contain the totally symmetric irrep.

a nontrivial task, especially when one hopes to also achieve maximum performance on the available computing resources.

Tensor contraction libraries are among the core components, together with molecular integral evaluation engines, of quantum chemistry programs. On modern distributed-memory parallel computing systems, the efficiency of a tensor contraction hinges on minimization of communication overhead among nodes. Tensor libraries should ideally provide a number of key performance capabilities [11]:

1. They must be able to accommodate arbitrary tensors, with storage strategies that minimize the need for noncontiguous data movement operations, such as transpositions and sortings.
2. They should handle arbitrary contractions between tensors using optimized and parallelizable computational kernels.
3. They should transparently and adaptively manage memory usage by processing tensors in chunks as required by the available amounts of core memory and disk space.
4. The should leverage all the symmetries that the quantum chemical theory at hand enforces, such as permutational, spin, and point group, which can lead to significant savings in terms of computational time and storage, as discussed in the previous sections.

Tensor library developers attempting to fulfill this list of requirements also strive in keeping the inherent complexities of the code properly encapsulated from the end-users of their tensor library: the design of the applications programmer interface (API) is extremely important. At a glance, many of the existing implementations conform to the so-called *transpose-transpose-GEMM-transpose* (TTGT) paradigm [12] such as those described in Section 2 above and are thus structured according to the following scheme:

- A low-level foundation that specifies the layout of the tensor data structure in memory and the basic operations on such a data structure, such as allocation, deallocation, assignment, element access, and arithmetic operations. This low-level layer, being "closer to the metal" will also take care of the chunking operations. The latter will be discussed shortly.
- A middle-level layer that handles contractions. In the TTGT paradigm, one will resort to optimized BLAS-like libraries for matrix-matrix and matrix-vector multiplication.
- A high-level layer that is ultimately intended as the library API for user consumption. This layer should allow for easy creation and manipulation of tensors and the expression in code of the working equations of the

theory in a form that is as close as possible to the derived equations. This task is easily achieved with the definition and implementation of domain-specific languages (DSLs). Automatic code generators [13, 14] might also be part of the high-level layer, but their discussion is beyond the scope of this article.

We emphasize that this design overview does not imply that there exists a one-size-fits-all approach to the problem. Whereas the basic requirements for a library will largely remain unchanged, the target problems (small and medium vs large) and architectures (single- vs multiple-node, homogeneous vs heterogeneous) will perforce guide the implementation choices.

In many medium-sized applications on single-node, homogeneous architectures, the available core memory might be exceeded by the required tensors. An efficient library must be able to process tensors in chunks and switch between disk-based and memory-based algorithms with little to no additional overhead. For this class of problems the existing implementations fall in the TTGT paradigm: the cost of transposition/sorting is still negligible with respect to that for matrix multiplications. Storage schemes where symmetry packing is easily implemented are favored, as they afford significant savings in the I/O operations for disk-based algorithms.

The direct product decomposition (DPD) approach [9, 10] (components of which are described in detail in Section 3.3) was devised to enforce point group symmetry in the solution of the coupled cluster equations. The storage format emphasizes space savings: only nonzero, nonredundant symmetry blocks of amplitudes, integrals, and intermediates will be stored. This will incur transposition/sorting overhead during some of the contractions, but it still is very well suited for medium-sized applications, especially if coupled to disk-based storage and chunking strategies that overlap disk I/O with matrix multiplies to hide latency.

The tensor contraction engine [15] (TCE) is another notable example of the TTGT paradigm. The TCE generates contraction code to be further compiled into an executable program. An operation tree is generated based on the derived equations, intermediates are identified, and the optimal contraction patterns are chosen based on heuristic considerations. More recently Epifanovsky et al. have described the `libtensor` library which offers an implementation of the TTGT paradigm with an easy to understand high-level API for increased programmers productivity [16, 17]. The work of Kaliman and Krylov on `libxm` [18] builds upon `libtensor`, to allow the use of GPUs and disk-based algorithms. Many of the more recent implementations of tensor libraries use C++, the

multiparadigm programming language that makes it possible to achieve performance without sacrificing expressiveness. Template programming is one of the essential cornerstones of C++, since it enables static polymorphism, compile-time computation, and laziness [19]. For example, with static polymorphism it is possible to have functions for single-precision and double-precision floating point numbers without code duplication. This technology is proving instrumental in reappraising the assumed necessity of double-precision arithmetic [20] and will be a necessary tool for any attempt at mixed-precision implementations.

For multiple-node and/or heterogeneous architectures, developers strive to minimize communication overhead and load imbalances between the different processing units. Thus, more often than not, transposition and sorting become bottlenecks as they require slow internode communication. Authors of this class of libraries emphasize storage formats that maximize data locality to facilitate distribution and reduce message-passing. Understandably, such libraries show greater complexity: assumptions on the layout of the computing facilities need to be implicitly or explicitly coded in the various algorithms.

The Cyclops tensor framework [21] (CTF) is designed to leverage multiple-node architectures within the TTGT paradigm. This requires carefully tuned storage patterns so that transposition/sorting will not become a communication bottleneck. The library provides an expressive DSL that lets users write code that maps one-to-one to derived equations. Under the hood, the tensors are distributed according to a cyclic decomposition, and structured communication patterns orchestrate the tensor contractions. The cyclic decomposition was adopted to preserve symmetry packing of the tensor in each subtensor while minimizing the need for padding data. The resulting decomposition is regular enough to allow the use of BLAS-like subroutines on such distributed blocks.

The ExaTensor library is a relatively new addition to the landscape of tensor contraction libraries for quantum many-body theories [22]. The library is written in the Fortran programming language and explicitly targets multiple-node, heterogeneous computing environments. The domain-specific virtual processor (DSVP) concept is the core abstraction of ExaTensor: a software processor architecture that is able to execute domain-specific primitives, such as tensor contractions, natively on the available hardware. This is an idea reminiscent of the super instruction architecture language (SIAL) framework [23–25], and strives for a task-based, rather than data-based, parallelism strategy.

Ultimately, as the sizes of chemical problems and computing architectures reach extreme scales, the cost of transposition/sorting will adversely affect performance. Furthermore, the emergence of local quantum many-body theories—which take advantage of the short-sightedness of the Coulomb two-electron interaction to achieve reduced scaling—poses additional challenges to the effectiveness of the TTGT paradigm. With locality, sparsity patterns emerge that are less regular than those imposed by the symmetries we have so far discussed.

The TBLIS project aims at completely avoiding transposition/sorting in tensor contractions [12]. Within TBLIS, tensors are stored as matrices, but according to a logical mapping that is neither column-major nor row-major. This is termed block-scatter-matrix (BSM) format and the BSM tensor contraction (BSMTC) is its related contraction algorithm. As the name suggests, TBLIS is related to the BLAS-like Library Instantiation Software (BLIS) framework [26].

TiledArray offers a novel approach to the efficient exploitation of sparsity and targets multiple-node architectures [27–29]. The basic data type is the distributed array, which is a collection of tiles, for dense or sparse tensors. The implementation is templated C++, and different underlying scalar data types are thus automatically supported. The library can also accommodate more general sparsity patterns, such as the clustered low-rank [30] (CLR) format, reminiscent of H-matrices.

We also note that most of what has been discussed above can be adapted, with minor modifications, to relativistic electronic structure models in the Kramers unrestricted formalism [31, 32]. Indeed, when the time-reversal symmetry between large and small component of the 4-spinors is relaxed, the working equations of relativistic coupled cluster theory, for example, are the same as in the nonrelativistic spin-orbital case, allowing for efficient reuse of much of the same infrastructure. Recently Shee et al. have presented a new implementation of relativistic ground- [33] and excited-state [33] coupled cluster methods based on an extension of the DPD format [9, 10] to double group symmetry and complex algebra.

5. Conclusions and prospectus

The quest for the optimum implementation of tensor contraction algorithms will likely keep researchers occupied for many years to come. The efficient use of existing computer architectures, programmers' productivity/time, and expansion of the domain of applicability of quantum

many-body methods pose important challenges. With heterogeneous computing environments gaining importance, careful hand-optimization of computational kernels could prove ineffective. Data-based approaches to systematic optimization, notably based on machine learning, could become the frontier of performance tuning.

Acknowledgments

This research was supported by grants (CHE-1900420 and ACI-1450169) from the U.S. National Science Foundation and a CCS grant from the U.S. Department of Energy. R.D. R was supported by the Research Council of Norway through its Centres of Excellence scheme, project number 262695, and through its Mobility Grant scheme, project number 261873. The authors also acknowledge Advanced Research Computing at Virginia Tech for providing computational resources and technical support that have contributed to the results reported within the paper.

References

1. Matthews, D. A.; Gauss, J.; Stanton, J. F. Revisitation of Nonorthogonal Spin Adaptation in Coupled Cluster Theory. *J. Chem. Theory Comput.* **2013**, *9*(6), 2567–2572. https://doi.org/10.1021/ct301024v.
2. Matthews, D. A.; Stanton, J. F. Non-Orthogonal Spin-Adaptation of Coupled Cluster Methods: A New Implementation of Methods Including Quadruple Excitations. *J. Chem. Phys.* **2015**, *142*(6), 064108. https://doi.org/10.1063/1.4907278.
3. Cotton, F. A. *Chemical Applications of Group Theory*. Wiley: New York, **1990**.
4. Furche, F.; Ahlrichs, R.; Hättig, C.; Klopper, W.; Sierka, M.; Weigend, F. Turbomole. *WIREs Comput. Mol. Sci.* **2014**, *4*(2), 91–100. https://doi.org/10.1002/wcms.1162.
5. Almlöf, J. *USIP Report 74–29*. University of Stockholm, **1974**, Tech. Rep.
6. Almlöf, J. *The MOLECULE Integral Program, Report 74–09*. Tech. Rep. **1974**.
7. Aidas, K.; Angeli, C.; Bak, K. L.; Bakken, V.; Bast, R.; Boman, L.; Christiansen, O.; Cimiraglia, R.; Coriani, S.; Dahle, P.; Dalskov, E. K.; Ekström, U.; Enevoldsen, T.; Eriksen, J. J.; Ettenhuber, P.; Fernández, B.; Ferrighi, L.; Fliegl, H.; Frediani, L.; Hald, K.; Halkier, A.; Hättig, C.; Heiberg, H.; Helgaker, T.; Hennum, A. C.; Hettema, H.; Hjertenæs, E.; Høst, S.; Høyvik, I. M.; Iozzi, M. F.; Jansík, B.; Jensen, H. J. A.; Jonsson, D.; Jørgensen, P.; Kauczor, J.; Kirpekar, S.; Kjærgaard, T.; Klopper, W.; Knecht, S.; Kobayashi, R.; Koch, H.; Kongsted, J.; Krapp, A.; Kristensen, K.; Ligabue, A.; Lutnæs, O. B.; Melo, J. I.; Mikkelsen, K. V.; Myhre, R. H.; Neiss, C.; Nielsen, C. B.; Norman, P.; Olsen, J.; Olsen, J. M. H.; Osted, A.; Packer, M. J.; Pawlowski, F.; Pedersen, T. B.; Provasi, P. F.; Reine, S.; Rinkevicius, Z.; Ruden, T. A.; Ruud, K.; Rybkin, V. V.; Sałek, P.; Samson, C. C. M.; de Merás, A. S.; Saue, T.; Sauer, S. P. A.; Schimmelpfennig, B.; Sneskov, K.; Steindal, A. H.; Sylvester-Hvid, K. O.; Taylor, P. R.; Teale, A. M.; Tellgren, E. I.; Tew, D. P.; Thorvaldsen, A. J.; Thøgersen, L.; Vahtras, O.; Watson, M. A.; Wilson, D. J. D.; Ziolkowski, M.; Agren, H. The Dalton quantum chemistry program system. *WIREs Comput. Mol. Sci.* **2014**, *4*(3), 269–284. https://doi.org/10.1002/wcms.1172.
8. Parrish, R. M.; Burns, L. A.; Smith, D. G. A.; Simmonett, A. C.; DePrince, A. E., III; Hohenstein, E. G.; Bozkaya, U.; Sokolov, A. Y.; Remigio, R. D.; Richard, R. M.; Gonthier, J. F.; James, A. M.; McAlexander, H. R.; Kumar, A.; Saitow, M.; Wang, X.; Pritchard, B. P.; Verma, P.; Schaefer, H. F., III; Patkowski, K.;

King, R. A.; Valeev, E. F.; Evangelista, F. A.; Turney, J. M.; Crawford, T. D.; Sherrill, C. D. Psi4 1.1: An Open-Source Electronic Structure Program Emphasizing Automation, Advanced Libraries, and Interoperability. *J. Chem. Theory Comp.* **2017**, *13*, 3185–3197.

9. Stanton, J. F.; Gauss, J.; Watts, J. D.; Bartlett, R. J. A Direct Product Decomposition Approach for Symmetry Exploitation in Many-Body Methods. I. Energy calculations. *J. Chem. Phys.* **1991**, *94*(6), 4334–4345. https://doi.org/10.1063/1.460620.
10. Gauss, J.; Stanton, J. F.; Bartlett, R. J. Coupled-Cluster Open-Shell Analytic Gradients: Implementation of the Direct Product Decomposition Approach in Energy Gradient Calculations. *J. Chem. Phys.* **1991**, *95*(4), 2623–2638. https://doi.org/10.1063/1.460915.
11. Windus, T. L.; Pople, J. A. Pinnacle: An Approach Toward Object Oriented Quantum Chemistry. *Int. J. Quantum Chem.* **1995**, *56*(S29), 485–495. https://doi.org/10.1002/qua.560560852.
12. Matthews, D. High-Performance Tensor Contraction Without Transposition. *SIAM J. Sci. Comput.* **2018**, *40*(1), C1–C24. https://doi.org/10.1137/16M108968X.
13. Janssen, C. L.; Schaefer, H. F. The Automated Solution of Second Quantization Equations With Applications to the Coupled-Cluster Approach. *Theor. Chem. Acc.* **1991**, *79*, 1.
14. Crawford, T. D.; Lee, T. J.; Schaefer, H. F. A New Spin-Restricted Triple Excitation Correction for Coupled Cluster Theory. *J. Chem. Phys.* **1997**, *107*, 7943–7950.
15. Hirata, S. Tensor Contraction Engine: Abstraction and Automated Parallel Implementation of Configuration-Interaction, Coupled-Cluster, and Many-Body Perturbation Theories. *J. Phys. Chem. A* **2003**, *107*(46), 9887–9897. https://doi.org/10.1021/jp034596z.
16. Epifanovsky, E.; Wormit, M.; Kuś, T.; Landau, A.; Zuev, D.; Khistyaev, K.; Manohar, P.; Kaliman, I.; Dreuw, A.; Krylov, A. I. New Implementation of High-Level Correlated Methods Using a General Block Tensor Library for High-Performance Electronic Structure Calculations. *J. Comput. Chem.* **2013**, *34*(26), 2293–2309. https://doi.org/10.1002/jcc.23377.
17. Ibrahim, K. Z.; Williams, S. W.; Epifanovsky, E.; Krylov, A. I. Analysis and Tuning ofLibtensor Framework on Multicore Architectures. *2014 21st International Conference on High Performance Computing (HiPC)*. **2014**;; pp 1–10. https://doi.org/10.1109/HiPC.2014.7116881.
18. Kaliman, I. A.; Krylov, A. I. New Algorithm for Tensor Contractions on Multi-Core CPUs, GPUs, and Accelerators Enables CCSD and EOM-CCSD Calculations With Over 1000 Basis Functions on a Single Compute Node. *J. Comput. Chem.* **2017**, *38*(11), 842–853. https://doi.org/10.1002/jcc.24713.
19. Vandevoorde, D.; Josuttis, N. M.; Gregor, D. *C++ Templates: The Complete Guide*, 2nd ed.; Addison-Wesley Professional, **2017**.
20. Pokhilko, P.; Epifanovsky, E.; Krylov, A. I. Double Precision Is Not Needed for Many-Body Calculations: Emergent Conventional Wisdom. *J. Chem. Theory Comput.* **2018**, *14*(8), 4088–4096. https://doi.org/10.1021/acs.jctc.8b00321.
21. Solomonik, E.; Matthews, D.; Hammond, J. R.; Stanton, J. F.; Demmel, J. A Massively Parallel Tensor Contraction Framework for Coupled-Cluster Computations. *J. Parallel Distrib. Comput.* **2014**, *74*(12), 3176–3190. https://doi.org/10.1016/j.jpdc.2014.06.002.
22. Lyakh, D. I. Domain-Specific Virtual Processors as a Portable Programming and Execution Model for Parallel Computational Workloads on Modern Heterogeneous High-Performance Computing Architectures. *Int. J. Quantum Chem.* **2019**, *119*(12), e25926. https://doi.org/10.1002/qua.25926.

23. Lotrich, V.; Flocke, N.; Ponton, M.; Yau, A. D.; Perera, A.; Deumens, E.; Bartlett, R. J. Parallel Implementation of Electronic Structure Energy, Gradient, and Hessian Calculations. *J. Chem. Phys.* **2008**, *128*(19), 194104. https://doi.org/10.1063/1.2920482.
24. Lotrich, V.; Flocke, N.; Ponton, M.; Sanders, B. A.; Deumens, E.; Bartlett, R. J.; Perera, A. An Infrastructure for Scalable and Portable Parallel Programs for Computational Chemistry. *Proceedings of the 23rd International Conference on Supercomputing*; ACM, **2009**, pp 523–524. https://doi.org/10.1145/1542275.1542361.
25. Lotrich, V. F.; Ponton, J. M.; Perera, A. S.; Deumens, E.; Bartlett, R. J.; Sanders, B. A. Super Instruction Architecture of Petascale Electronic Structure Software: The Story. *Mol. Phys.* **2010**, *108*(21–23), 3323–3330. https://doi.org/10.1080/00268976.2010.512566.
26. Van Zee, F. G. van de Geijn, R. A.. BLIS: A Framework for Rapidly Instantiating BLASFunctionality. *ACM Trans. Math. Softw.* **2015**, *41*(3), 14:1–14:33. https://doi.org/10.1145/2764454.
27. Calvin, J. A.; Valeev, E. F. *Task-Based Algorithm for Matrix Multiplication: A Step Towards Block-Sparse Tensor Computing*. http://arxiv.org/abs/1504.05046; **2015**.
28. Calvin, J. A.; Lewis, C. A.; Valeev, E. F. Scalable Task-Based Algorithm for Multiplication of Block-Rank-Sparse Matrices. *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*; ACM, **2015**, p 4. https://doi.org/10.1145/2833179.2833186.
29. Peng, C.; Calvin, J. A.; Pavošević, F.; Zhang, J.; Valeev, E. F. Massively Parallel Implementation of Explicitly Correlated Coupled-Cluster Singles and Doubles Using TiledArray Framework. *J. Phys. Chem. A* **2016**, *120*(51), 10231–10244. https://doi.org/10.1021/acs.jpca.6b10150.
30. Lewis, C. A.; Calvin, J. A.; Valeev, E. F. Clustered Low-Rank Tensor Format: Introduction and Application to Fast Construction of Hartree-Fock Exchange. *J. Chem. Theory Comput.* **2016**, *12*(12), 5868–5880. https://doi.org/10.1021/acs.jctc.6b00884.
31. Visscher, L.; Dyall, K. G.; Lee, T. J. Kramers-Restricted Closed-Shell CCSD theory. *Int. J. Quantum Chem.* **1995**, *56*(S29), 411–419. https://doi.org/10.1002/qua.560560844.
32. Visscher, L.; Lee, T. J.; Dyall, K. G. Formulation and Implementation of a Relativistic Unrestricted Coupled-Cluster Method Including Noniterative Connected Triples. *J. Chem. Phys.* **1996**, *105*(19), 8769–8776. https://doi.org/10.1063/1.472655.
33. Shee, A.; Visscher, L.; Saue, T. Analytic One-Electron Properties at the 4-Component Relativistic Coupled Cluster Level With Inclusion of Spin-Orbit Coupling. *J. Chem. Phys.* **2016**, *145*(18), 184107. https://doi.org/10.1063/1.4966643.